Identifying Productivity Drivers by Modeling Work Units Using Partial Data

Todd L. GRAVES
Los Alamos National Laboratory
MS F600
Los Alamos, NM 87545

Audris MOCKUS
Bell Laboratories
263 Shuman Blvd
Naperville, IL 60566

We describe a new algorithm for estimating a model for an independent variable which is not directly observed, but which represents one set of marginal totals of a sparse nonnegative two-way table whose other margin and zero pattern are known.

The application which inspired the development of this algorithm arises in software engineering. It is desired to know what factors affect the effort required for a developer to make a change to the software: for instance, to identify difficult areas of the code, measure changes in the code difficulty through time, and evaluate the effectiveness of development tools. Unfortunately, measurements of effort for changes are not available in historical data. We model change effort using a developer's total monthly effort and information about which changes she worked on in each month. We illustrate a few specific applications of our tool, demonstrate that the algorithm is an instance of the EM algorithm, and present a simulation study which speaks well of the reliability of the results our algorithm produces. In short, this algorithm allows analysts to quantify the impact a promising software engineering tool or practice has on coding effort.

1. INTRODUCTION

A quantity particularly important to software development organizations is the cost of making a change to their software. In this paper we discuss the methodology and algorithms we have developed to assess which factors influence software change effort.

Large software systems are maintained using change management systems, which record information about all the changes programmers make to the software. These systems record a number of measurements on each change, such as its size and type, which one expects to be related to the effort required to make the change. However, change management data do not include measurements of the effort that is required to make a change, so that it is at first difficult to imagine estimating models for change effort. Effort estimates are often obtained for very large changes like new releases, but is important to study effort at the level of individual changes, because qualities affecting the effort of individual changes are often not estimable after aggregation. Therefore, our methodology makes use of two types of

TECHNOMETRICS, ???

known information: the total amount of effort a developer expends in a month, and the set of months during which the developer worked on a given change.

This gives rise to an ill-posed problem in which developers' monthly efforts represent the known column marginal totals of several two-way tables, and we also know which entries in these sparse tables are strictly positive. Also, we postulate the form of a model for the unobservable row marginal totals (amounts of effort for each change) as a function of several observed covariates. Our algorithm then iterates between improving the coefficients in the model for effort using imputed values of change effort, and adjusting the imputed values so that they are consistent with the known measurements of aggregated effort. We have called this algorithm MAIMED, for Model-Assisted Iterative Margin Estimation via Disaggregation. Uncertainty estimates for the model coefficients can then be obtained using the jackknife (Efron, 1982), in which each replication omits one of the two-way tables (developers). We anticipate that this methodology will be useful in other sorts of problems where individuals produce units of output, cost information is available only in units of time, and where the variables affecting productivity operate on the level of the output units.

This methodology has enormous potential for serving as a foundation for a variety of historical studies of software change effort. One example from this research is our finding that a subsystem of the code under study "decayed" in that changes to the code grew harder to make at a rate of 20% per year. We were also able to measure the extent to which changes which fix faults in software are more difficult than comparably sized additions of new functionality: fault fixes can be as much as 80% more difficult. Other findings include evidence that a development tool makes developers more productive, and confirmation that a collection of subsystems believed to be more difficult than others in fact required measurably more effort to make an average change.

We present recommendations on using the estimation method in practice, discuss alternative approaches and present a simulation study to confirm accuracy and study sensitivity to model misspecification and initial values.

After further discussion of the software application in this section, the next section defines the estimation algorithm. Section 3. demonstrates that MAIMED is an EM algorithm, and compares our approach to existing work. Section 4. describes three sample data analyses using these methods. Section 5. describes our simulation study, and Section 6. gives our conclusions. Appendix 1 describes the software used to perform the estimation.

1.1 Data sources for the software engineering application

Change history is a promising new tool for measuring and analyzing software evolution (Mockus *et al*, 1999). Change histories are automatically recorded by *source code control* and *change management systems* (see Section 1.2). Every change to every part of the source code is recorded by the change control system, providing a repository of software modifications which is large, detailed, and uniform over time. Obtaining data for software engineering research relevant to an industrial environment can be difficult and costly; change management data are collected automatically.

In our studies, the data represent changes to the source code of 5ESS™ (Martersteck and Spencer, 1985), which is a large, distributed, high availability software product. The product was developed over two decades, has tens of millions of lines of source code, and has been changed several hundred thousand times. The source code is mostly written in the C programming language, augmented by the Specification and Description Language (SDL). Tools used to process and analyze software change data are described in Mockus *et al*, 1999.

The change management (CM) data include information on each change made to the code, its size and content, submission time, and developer. Also available are financial support system (FSS) data, which record amounts of effort spent each month by each developer. Because developers tend to work on multiple changes during most months, and because 14 percent of changes start and end in different months, it is impossible to recover effort measurements for individual changes exactly from these FSS data. In fact, developers' monthly efforts rarely stray far from their natural value of one month, so in most applications we have used monthly efforts identically equal to one in place of the FSS data, which in any case are not always available.

1.2 Records in change management data

The extended change management system (ECMS; see, for example, Midha, 1997), and the source code control system (SCCS; Rochkind, 1975) are used to manage the source code. The ECMS groups atomic changes ("deltas") recorded by SCCS into logical changes referred to as Maintenance Requests (MRs). The open time of each MR is recorded in ECMS. Also available are records of the MR completion time; we use the time of the last delta in that MR. We used the textual abstract that the developer wrote when working on an MR to infer that change's purpose (Mockus and Votta, 1998). In addition to the three established reasons for changes (repairing faults, adding new functionality, and improving the structure of the code; see, for example, Swanson, 1976), we defined a class for changes that implement code inspection suggestions, since this class was easy to separate from others and had distinct size and interval properties (Mockus and Votta, 1998).

The SCCS database records each delta as a tuple including the actual source code that was changed, login name of the developer, MR number, date, time, numbers of lines added, deleted, and unchanged.

2. METHODOLOGY

In this section we discuss our procedure for estimating models for change effort. Consider the work history corresponding to a single developer d. These quantities may be thought of as a non-negative two-way table,

$$\{Y_{ijd} \geq 0 : 1 \leq i \leq M, 1 \leq j \leq N\}.$$

Here i ranges over the M work units (here, changes to the code), and j ranges over the time periods (here, months) over which effort data are available. Known are the zero pattern

$$\{(i,j) : Y_{ijd} = 0\},$$

since change data record when work on a change begins and ends, and the column sums

$$Y_{\cdot jd} = \sum_{i=1}^{M} Y_{ijd}.$$

In the software engineering problem, the unobservable quantities of interest are the amounts of effort required for each change, represented by the row sums $Y_{i\cdot d} = \sum_{j=1}^{N} Y_{ijd}$. A statistical model for these can be expressed as

$$L(E(Y_{i\cdot d})) = X'_{id}\beta, \tag{1}$$

where X_{id} is a vector of covariates predicting change effort, β is a vector of unknown parameters whose estimation is the goal of the procedure, $'$ denotes transpose, and L is a link function. (Since the quantities to be predicted are positive, generalized linear models – see McCullagh and Nelder, 1989 – which lead to multiplicative models are appropriate).

2.1 Fitting algorithm

The estimation procedure is inspired by iterative proportional fitting (see Deming and Stephan, 1940, and Darroch and Ratcliff, 1972), in which initial values of cell probabilities in a table are iteratively adjusted to make them consistent with known marginal totals. Throughout the algorithm, estimates for the cells which are known to be zero remain zero, so update formulas given below refer only to positive cell values. We first generate initial values for the cells in the two-way tables. A natural approach is to divide each column total evenly across all the nonzero entries in that column: define $s_d(j) = |\{k : Y_{kjd} > 0\}|$ to be the number of strictly positive entries in the jth column of the table. If $Y_{ijd} > 0$, let

$$Y_{ijd}(0) = s_d(j)^{-1} Y_{\cdot jd}. \tag{2}$$

Here we have denoted by $Y_{ijd}(t)$ the estimate of Y_{ijd} after the t-th iteration of the algorithm; it will be necessary to allow the argument t to take on fractional values because the cell values are updated twice within each iteration. Alternative initialization methods are considered in Section 5.4.

The following steps are then repeated until convergence.

1. Generate row sums from the current estimate of the cell values,

$$Y_{i\cdot d}(t) = \sum_{j=1}^{N} Y_{ijd}(t).$$

2. Fit the statistical model given by (1), using each of the tables (one for each developer $d = 1, 2, \ldots, D$). This model generates regression coefficients $\beta(t)$ and fitted values $\hat{Y}_{i \cdot d}(t)$.

3. Rescale the cell values so that the row sums are equal to the model's fitted values:

$$Y_{ijd}(t+1/2) = Y_{ijd}(t)\{\sum_{\ell=1}^{N} Y_{i\ell d}(t)\}^{-1}\hat{Y}_{i\cdot d}(t).$$

4. Finally rescale the cell values so that they agree with the known column sums:

$$Y_{ijd}(t+1) = Y_{ijd}(t+1/2)\{\sum_{k=1}^{M} Y_{kjd}(t+1/2)\}^{-1}Y_{\cdot jd}.$$

Convergence is obtained when the improvement in the error measure in the model fitting is negligible. As a consequence of the identification of MAIMED as an EM algorithm, the error measure decreases at each iteration, and in practice, we have found that convergence is faster with a Poisson generalized linear model with a log link (i.e. a multiplicative model) than for an additive model.

At convergence, the algorithm reports the coefficients estimated in the model fitting during the last iteration.

2.2 Jackknife estimates of parameter uncertainty

Since these regression coefficients are obtained using an iterative algorithm and a number of fitted regressions, it is necessary to measure their uncertainties using a nonparametric method which takes the estimation procedure into account. The jackknife (see, for instance, Efron, 1982) supplies a method of estimating uncertainties in estimated parameters by rerunning the estimation algorithm once for each data point in the sample, each time leaving one of the data points out. A collection of estimated parameters results, and the degree of difference between these parameters is related to the sampling variability in the overall estimate. For example, suppose that the statistic of interest is denoted by $\hat{\theta}$, which was computed from a dataset of size n. What is required is an estimate of the standard error of $\hat{\theta}$. For $i = 1, 2, \ldots, n$, compute $\hat{\theta}_{(i)}$ using the data set with the ith observation deleted. Set $\hat{\theta}_{(\cdot)} = n^{-1}\sum_{i=1}^{n}\hat{\theta}_{(i)}$. The jackknife estimate of standard error of $\hat{\theta}$ is

$$\left\{\frac{n-1}{n}\sum_{i=1}^{n}(\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2\right\}^{1/2}.$$

An alternative to the jackknife is the bootstrap (for which see Efron and Tibshirani, 1993, for example). Owing to the time-consuming iterative algorithm, we preferred the jackknife since it required only one additional run of the algorithm for each of the developers. For the purposes of resampling, we have only one datapoint for each developer, because omitting some of a developer's changes leaves some months in which the total effort of the changes is less than the observed monthly effort. Omitting some months will break up changes whose lifetimes span those months, making it impossible to define the total effort for such a change.

The fact that MAIMED is an EM algorithm suggests standard error computations related to the observed information matrix as an alternative; see Chapter 4 of McLachlan and Krishnan (1997).

3. RELATED WORK

In this section we compare the MAIMED algorithm to previous work on related problems. First, we discuss the relationships of MAIMED with the EM algorithm and with iterative proportional fitting. Next, we discuss the possibility of constructing a more realistic statistical model, including distributions for quantities which MAIMED leaves unspecified, and demonstrate challenges posed by realistic problem formulations with such models.

3.1 EM Algorithm

As pointed out by this paper's referees, MAIMED is an instance of the EM algorithm for the Poisson likelihood. For $1 \leq i \leq M$ and a fixed developer d, define $r_d(i)$ to be the cardinality of the set $\{j : Y_{ijd} > 0\}$. Analogously define $s_d(j)$ to be the cardinality of $\{i : Y_{ijd} > 0\}$. We will present the argument as if there were only one developer d; the extension to multiple developers is immediate. Write I_d to be the set of pairs (i, j) such that $Y_{ijd} > 0$. Assume independent Poisson likelihood for the Y_{ijd} with means μ_{ijd}, where $\mu_{ijd} = 0$ for all $(i, j) \notin I_d$, and where

$$\mu_{i \cdot d} = \sum_j \mu_{ijd} = \sum_{k=1}^{p} x_{ik} \beta_k.$$

We reparametrize the likelihood in terms of $\mu_{i \cdot d}$, $p_{ijd} = \mu_{ijd}/\mu_{i \cdot d}, j = 1, \ldots, r_d(i) - 1$. The p_{ijd}'s are allowed to vary so that they can be estimated independently of β.

Given the complete data Y_{ijd}, the likelihood for μ_{ijd} is

$$L(\beta) = \sum_{(i,j) \in I_d} \{Y_{ijd} \log \mu_{ijd} - \mu_{ijd}\}.$$

However, this expression is equal to

$$\sum_{i=1}^{M} \{Y_{i \cdot d} \log \mu_{i \cdot d} - \mu_{i \cdot d}\} + \sum_{(i,j) \in I_d} Y_{ijd} \log(\mu_{ijd}/\mu_{i \cdot d}). \tag{3}$$

The second term in this sum is expressed purely in terms of p_{ijd}s and does not depend on the unknown parameters β, so that maximum likelihood estimation of β is done based only on the first term and is equivalent to maximum likelihood estimation based only on the $Y_{i \cdot d}$'s, which is how β is estimated in MAIMED. It also follows from (3) that the maximum likelihood estimates of p_{ijd} satisfy

$$\hat{p}_{ijd} = Y_{ijd} Y_{i \cdot d}^{-1}.$$

Identifying $\hat{p}_{ijd}\hat{\mu}_{i\cdot d}$ with $Y_{ijd}(t+1/2)$ in the specification of the MAIMED algorithm, one sees that Step 3 is in fact an M step of an EM algorithm. Further, it follows from the conditional distribution of an individual Poisson variable given the sum of it and other independent Poisson variables, that Step 4, namely

$$Y_{ijd}(t) = \hat{\mu}_{ijd}\{\sum_{k=1}^{M} \hat{\mu}_{kjd}\}^{-1} Y_{\cdot jd},$$

constitutes an expectation step.

The fact that MAIMED is an instance of an EM algorithm implies applicability of the general convergence theorems described in, e.g., Chapter 3 of McLachlan and Krishnan (1997). For example, MAIMED can be guaranteed to converge to some limit, so long as the implementation of generalized linear model fitting succeeds in increasing the likelihood.

Unfortunately, the Poisson likelihood is unrealistic in the context in which we apply MAIMED. Some of the objections are that the Y_{ijd}'s are not integer valued, and that independent Poisson variables will not have the property that total monthly efforts by developers tend to be very close to one unit of effort. This led us to investigate the behavior and robustness of MAIMED for the distributions that occur in practice, using the simulations in Section 5.. Additionally, some of these objections may be addressed using similar arguments to those related to quasi-likelihood in McCullagh and Nelder (1989), in which it is argued that likelihood functions are useful for inference if the data distribution agrees with the first two moments of the likelihood. Here, that would mean that variances of monthly efforts were proportional to their means. In our problem, variances are in fact proportional to squared means, but Poisson models are still more appropriate because they in effect penalize errors proportionally to their magnitude. Gamma models, on the other hand, approximately penalize percentage error, and multiplicative errors are more serious in the effort problem for large effort values. See Appendix 2 for details of this argument.

3.2 Relationship to iterative proportional fitting

Iterative proportional fitting (IPF; Deming and Stephan, 1940) is frequently used for estimating cell values in a multi-way table of probabilities when marginal totals are known. It also requires some sort of information about the table's association structure to be used in generating initial values for the iteration procedure; this information can take the form of a sample from the population. IPF alternates (in the two-way case) between rescaling the rows so that the cells sum to the correct row totals, and performing the same operation on the columns, until a normally rapid convergence. A related use of IPF involves fitting log-linear models to frequency tables (see Darroch and Ratcliff, 1972).

Our problem differs from existing domains of application of IPF in that we allow only one of the margins to be fixed, and modify the other margin in accordance with repeated fits of a regression model. In our application, one of the margins changes with each iteration. The row-column association structure that corresponds to MAIMED is also difficult to describe explicitly. The known zero pattern of the sparse tables we work with determines a large component of the association structure. No information exists to specify the remainder explicitly, unless one wishes to assume row-column independence as much as is possible. The independence assumption, however, is unrealistic, and in fact prevents the iterative improvement of the regression model which is the key to MAIMED. The EM formulation of the problem assumes that cell values are independent, and when we assume that the sums of the means within a row follow a parametric model and work conditionally on the column sums, this implies that the model assumes an association structure. This structure, however, cannot be specified in an informative way.

Simulations reported in Section 5. demonstrate that the limit attained by MAIMED depends very weakly on which of two reasonable methods for generating initial values we use. For this reason we argue that the assumptions one makes on the association structure between rows and columns are not important in the sparse tables of software effort data, and that MAIMED does not suffer from making no such explicit assumptions.

3.3 Alternative approaches

The fact that the effort modeling problem involves missing data makes it tempting to try to implement analyses of models arguably more realistic than the model under which MAIMED is EM. We indicate some difficulties with these approaches. The missing data mechanism for the quantity of interest, MR effort, is of an unusual form: in most cases this quantity is unobserved with probability one, but no other MRs were worked on in the months of the MR's lifetime, the effort for the MR is available.

We outline an attempt at a full and realistic probability model for all the data involved in the problem, after which the analyst may in principle employ familiar techniques such as the EM algorithm (see above) or data augmentation (see Tanner, 1993, for either). Here we describe such a model which is simple yet not unrealistic but which is challenging to implement. First, condition on the set of changes assigned to each developer and the times at which they began work on them[1]. Each change's effort is modeled as having a gamma distribution with mean depending on its type, size, developer assigned to it, and any other important factors (Section 4.1.1).

Once efforts and open times have been assigned to all changes, close times can be computed if one specifies how developers divide up their time across changes they work on simultaneously. Let change i by developer d have "urgency" u_{id}, where a developer divides up her time among changes proportionally to their urgencies. The urgencies will have distributions that depend on covariates: bug fixes will typically be more urgent than new feature additions. Knowing all changes' open dates, close dates, and urgencies allows the analyst to reconstruct change effort values. Statistical inference for this problem, in the context of either the EM algorithm or a Bayesian analysis, requires imputing values of urgencies. The distribution of urgencies will be conditional on open dates, close dates, the covariates which drive urgencies and efforts, and the current estimates of the unknown parameters in the distribution of urgencies and in the regression relationships for urgencies and efforts. Sampling from a distribution of the u's conditional on all these values is at best much more difficult than MAIMED: writing down the joint distribution of the urgencies and the efforts is straightforward. For a single developer d, the joint distribution of parameters, covariates, urgencies, and change efforts is

$$\pi(\beta,\gamma)f(\mathbf{x})\prod_i\{g(Y_{i\cdot jd}|\mathbf{x_i},\beta)h(u_{id}|\mathbf{x_i},\gamma)\},$$

where π is a prior density for the regression coefficients β and γ, f is the density of the covariates $\mathbf{x}$, g is the gamma density with mean and variance given by the covariates and the regression coefficients β, and h is a density for the urgencies. However, to sample from the distribution of urgencies given close times, one must change variables. This is problematic, since a change's close time is determined by the urgencies of changes the developer works on concurrently, a set of changes which is not computable until previous close times are known. Further, this approach relies on an assumption which may not be realistic: that urgencies do not change through time. The computational difficulties of this approach did not match our general goal of producing a simple to use method that could be routinely applied by a practitioner.

Alternatively, one can use multiple imputation (Rubin (1987, 1996)). The difficulty with this approach is to generate multiple complete tables using a sensible distribution for cell values. Since that distribution is based on the parameters that we want to estimate using the complete table, multiple imputations cannot be directly applied in this problem.

3.4 Relationship to network tomography

The problem of estimating network traffic from edge counts, considered in Vardi (1996), has some similarities with the present problem. There, numbers of traversals of each edge in a network are available, but the traffic for complete paths (sequences of connected edges) is of interest. Both problems deal with nonnegative quantities, and in each case linear combinations of quantities of interest are observed, since edge traffic is the sum over all the paths that traverse that edge.

In the network problem it is desired to recover a vector X given a vector Y of sums of the elements of X, satisfying the equation $Y = AX$, for some known matrix of zeroes and ones A. The effort problem adds another layer of difficulty to the network problem because there are more X's than Y's, and the positive values of A need not be ones and are unknown. In the effort estimation problem we seek a collection of sums $X = A1$ of unobservable quantities A, given another set of sums $Y = 1'A$ and the zero pattern of the matrix A. Auxiliary modeling information relates to the desired collection of sums, while in the network problem, it is the unsummed values which can potentially be modeled.

4. DATA ANALYSIS RESULTS

This section contains results of data analyses we have performed using the MAIMED methodology. These include an investigation of the existence of "code decay" (the deterioration of the maintainability of software over time), a study of the benefits imparted by a development tool, and a comparison of several subsystems of the software which folklore held to be either especially difficult or especially easy to modify. First, we discuss some hints for successful use of MAIMED.

4.1 Practical issues

There a number of practical issues when modeling change effort. Some are typical to all statistical problems, some specific to software engineering, and some specific to the MAIMED fitting procedure. Here we omit the traditional validity checks in regression analysis such as collinearity among predictors and dwell mostly on issues related to software engineering and the MAIMED procedure. Two particularly important issues are including essential predictors in the model and choosing subsets of the data.

4.1.1 Which variables are needed for the model in software applications?

In our studies of software data, we have found that several variables should always be included in the regression models and that their effects on effort are always present. One can include other variables to test whether they have important effects.

First among variables that we recommend always including in the model is a developer effect. Other studies (for example, Curtis, 1981) have found substantial variation in developer productivity. Even when we have not found significant differences and even though we do not believe that estimated development coefficients constitute a reliable method of rating developers, we have left developer coefficients in the model. The interpretation of estimated developer effects is problematic. Not only could differences appear because of differing developer abilities, but the seemingly less productive developer could be the expert on a particularly difficult area of the code, or that developer could have more extensive duties outside of writing code.

Naturally, the size of a change has a strong effect on the effort required to implement it. A large number of measures are available which measure something closely related to size, and the analyst should generally include exactly one of these in the model. We have typically chosen the number of atomic changes (deltas) that were part of the MR as the measure of the size of an MR. This measure seems to be slightly better for this purpose than other size measures like the total number of lines added and deleted in the MR. Another useful measure of size is the span of the change, i.e., the number of files it touches. A large span tends to increase the effort for a change even if other measures of size are constant.

We found that the purpose of the change (as estimated using the techniques of Mockus and Votta, 1998) also has a strong effect on the effort required to make a change. In most of our studies, changes which fix bugs are more difficult than comparably sized additions of new code. Difficulty of changes classified as "cleanup" varies across different parts of the code, while implementing suggestions from code inspections is easy.

4.1.2 Selecting subsets of developers

Another important problem when using MAIMED methodology is choosing a set of developers whose changes to include in the analysis. The 5ESS™ has been changed by a large number of developers, and we recommend restricting attention to a subset of these, with the subset chosen in order to yield the sharpest possible estimates of the parameters of interest.

Variability in project size and developer capability and experience are the largest sources of variability in software development (see, for example, Curtis, 1981). The effects of tools and process are often smaller by an order of magnitude. Therefore it is critical to avoid confounding effects of interest with developers, and it is useful to try to minimize variation between developers. For these reasons, one should select developers who make similar numbers of changes. When studying tool effects (see Section 4.2.2), one chooses developers who make similar number of changes with and without the tool. When estimating code-base effects (see Section 4.2.3), it is important to select developers who make similar numbers of changes on each type of product.

Given the considerable size of the version history data available, these tasks are often easy: in the tool effect study we selected developers who made between 300 and 500 MRs in the six year period between 1990 and 1995 and had similar numbers (more than 40) of MRs done with and without the tool.

We used relatively large samples of MRs in all three examples below. When comparing subsystems, we used 6985 MRs over 72 months to estimate 18 developer coefficients and 6 additional parameters. When analyzing the tool benefits, we used 3438 MRs over 72 months for 9 developers and five additional parameters.

4.2 Results of Data Analyses

The specific model we fit in the modeling stage of the algorithm was a generalized linear model (McCullagh and Nelder, 1989) of effort. MR effort was modeled as having a Poisson distribution with mean given below.[2]

$$E(Y_{i \cdot d}) = \alpha_d \times \beta_{Type(i,d)} \times \text{Size}_{id}^{\gamma} \times \theta_{\text{Cause}(i,d)}. \tag{4}$$

Here $Y_{i \cdot d}$ is the effort required for the ith change by the dth developer, α_d is the dth developer effect, Type(i, d), Size$_{id}$, and Cause(i, d) are respectively the type, size, and indicator of the presence of "Cause" for the ith change by the dth developer. The α's, β's, θ's, and γ are estimated using regression. Note that this multiplicative model is a generalized linear model with logarithmic link function. The covariate "Cause" is different in each of three studies summarized below. The decay study (Graves and Mockus, 1998) investigated the effects of calendar time, the tool usage study considered cost savings from using a tool, and the subsystem study investigated effects of different source code bases.

4.2.1 Code Decay

Beyond developer, size, and type of change, another interesting measurement on a change which we have found to be a significant contributor to the change's difficulty is the date of the change (Graves and Mockus, 1998). We were interested to see if there was evidence that the code was getting harder to change, or *decaying*, as discussed in Belady and Lehman (1976), Parnas (1994), and Eick *et al* (1999). There was statistically significant evidence of a decay effect: we estimated that in our data, a change begun a year later than an otherwise similar change would require 20% more effort than the earlier change. The fitted model was:

$$E(Y_{i \cdot d}) = \alpha_d \times \beta_{Type(i,d)} \times \text{Size}_{id}^{\gamma} \times \theta^{\text{Time(i,d)}}. \tag{5}$$

Here the estimated developer coefficients, the α's, varied by a factor of 2.7. The jackknife estimate of standard error (see Section 2.2) indicated that there was no statistically significant evidence that the developers were different from each other. Some investigators (e.g. Curtis, 1981) have found differences in developer productivity of a factor of ten to 100. A smaller ratio for our study is unsurprising, since we selected developers in a way which helps ensure that they are relatively uniform, by requiring that they completed large numbers of changes. Productivity ratios like these are in any case highly variable as they are based on the extreme developers in a set. Also, we believe that there are a number of tasks beyond coding (requirements, inspections, architecture, project management) that take substantial time from developers involved in such tasks. Consequently, the coding productivity α_d^{-1} does not necessarily reflect the total productivity or the overall value of the developer to an organization. It is promising to treat developer coefficients as random effects.

We defined β_{NEW}, the coefficient for changes that add new functionality, to be unity to make the problem identifiable, then estimated the remaining type coefficients to be 1.8 for fault fixes, 1.4 for restructuring and cleanup work, and 0.8 for inspection rework.

The value of γ, which is the power of the number of files changed which affects effort, we estimated to be 0.26, and its jackknife standard error estimate is 0.06. Since $\gamma < 1$, the difficulty of a change increases sub-linearly with the change's size. One might expect a effort to be nearly proportional to size (see, for example, the COCOMO model for effort prediction of Boehm, 1981). This expectation is reasonable in the context of large changes, such as entire projects, which contain mixtures of small changes of various types. When considering changes as small as MR's, a sublinear result should not be surprising. Large changes are often additions of new functionality and may be mostly straightforward, while small changes are often bug fixes, which may require large initial investments of time to find where to make the change.

We estimated θ to be 1.21, and its natural logarithm had a standard error of 0.07, so that a confidence interval for θ calculated by taking two standard errors in each direction of the estimated parameter is $1.21 \exp\{\pm 2 \times 0.07\} = (1.05, 1.39)$. This implies that a similar change became 5% to 39% harder to make in one year. (Intervals with other confidence levels can be obtained by changing the width from four standard errors to use other normal quantiles).

4.2.2 Tool usage effects

This section tests whether the use of a tool reduces the effort needed to make a subset of changes. The tool in this study was the Version Editor, which was designed to make it easier to edit software which is simultaneously developed in several different versions. It does this by hiding all lines of code which belong to versions the developer is not currently working on (for details see Atkins *et al* (1999)). The tool divided changes into three types:

1. a "control" set of changes which modified only files which had no versioning information. Here usage of the tool should have had no effect. Roughly half of all the changes fell into this category; they will be denoted as CONTROL;
2. changes where the tool should have an effect (i.e. those modifying files with versioning information) and the tool was used (denoted as USED);
3. changes where tool should have an effect and it was not used (denoted as NOT).

We fit the model (6), estimated standard errors using the jackknife method, and obtained the following results, as summarized in Table 1.

$$E(Y_{i\cdot d}) = \alpha_d \times \beta_{Type(i,d)} \times \mathrm{Size}_{id}^{\gamma} \times \theta_{\mathrm{Tool}(i,d)}. \tag{6}$$

The penalty for failing to use the tool is the ratio of θ_{NOT} to θ_{USED}, which indicates an increase of about 36% in the effort required to complete an MR. (This coefficient was statistically significant at the 5% level). Restated, developers who used the tool to perform non-control changes could perform 36% more changes than if they had used the tool. At the same time, changes performed using the tool were of the same difficulty (requiring a statistically insignificant 7% increase in effort) as CONTROL changes (θ_{USED} versus $\theta_{CONTROL}$).

The type of a change was a significant predictor of the effort required to make it, as bug fixes were 26% more difficult than comparably sized additions of new functionality. Two other types of changes (improving the structure of the code and implementing code inspection suggestions) were both of comparable difficulty to adding new code.

The variable γ in Equation (4) was estimated to be 0.19. That is, the size of a change did not have a particularly strong effect on the effort required to make it.

4.2.3 Subsystem effects

In the last example we considered the influence of code-base on effort. The 5ESS™ switch (the product under discussion) is broken into a number of relatively independent subsystems (each several million lines of code) that belong to and are changed by separate organizations. This product is updated by adding new features or capabilities that can then be sold to customers. An average feature contains tens to hundreds of MRs.

A development manager identified six subsystems that tend to have higher cost per feature than another six similar sized subsystems. To identify whether the effort was higher even at the MR level, we chose 18 developers who made similar numbers of changes to high- and low-cost subsystems. We fit the model (7). The estimates are summarized in Table 2. Since exp(0.249) = 1.28, the results show that similar MRs take about 28% more effort in the high cost subsystems.

$$E(Y_{i \cdot d}) = \alpha_d \times \beta_{Type(i,d)} \times \text{Size}_{id}^{\gamma} \times \text{Files}_{id}^{\gamma_1} \times \theta_{\text{Low}(i,d)}. \tag{7}$$

In this example we used two measures of the size of a change, adding "Files," the number of files touched by an MR. The factor "LOW" indicates that it is a low-cost-per-feature subsystem.

5. SIMULATION

We conducted a simulation experiment to evaluate the MAIMED algorithm on a number of realistic examples. The goals of these simulations were to simulate the data as closely as possible to a real software environment, using distributions that occur in practice. We studied the effects of model misspecification.

The simulations consisted of several parts. First, we simulated data from a specified model and verified that MAIMED did an adequate job of recovering the true coefficients (Section 5.2). We also generated data for which developers' monthly efforts were not identically one, and verified that using ones in place of these values yielded conservative results (Section 5.3). Also, we confirmed that altering the method for choosing initial values for the iteration had a very small effect, so that there is not really much freedom for assumptions about association structure in the two-way tables (Section 5.4). Finally, we demonstrated that MAIMED continues to perform well in the presence of model misspecification, by fitting models without one of the essential variables, or with an extra variable which does not itself affect effort but which is correlated with one of the predictor variables (Section 5.5).

The simulation is in part a bootstrap analysis of the data. It begins with a model fit to a set of data. We used the model from the tool usage example (6) and the estimated coefficients from Table 1.

Each replication in the experiment consists of the following. We sample a set of developers with replacement. The set of developers in the simulated data consists of a random number of copies of each of the developers in the true data set. We then construct a collection of MRs for a synthetic developer by resampling MRs of the corresponding actual developer. More precisely, we do the following.

1. Resample a set of developers with replacement. The following steps are iterated over each developer in the resulting set.
2. Choose an MR at random from the developer's collection. This MR has associated with it a size, type, and other covariates (ignore its open or close time).
3. Derive the MR's open date by assuming it was opened at the time the MR opened previously to this MR for the same developer was closed (use time zero if this is the first MR for the developer).
4. Adjust the open date for this MR by generating a time shift using parametric bootstrap (for details see Section 5.1).
5. Compute the expected value of the effort for this synthetic MR using the covariates for the resampled MR and the estimated model.

6. Generate the effort for this MR according to the exponential distribution with the computed mean. The close date of this MR is then the open date, plus this effort value.

This simulation procedure preserves the correlations between all the variables in the model, except for time. In the simulations, the covariates and effort for MRs are independent of the covariates and effort of the previous MRs. Since the actual MRs do overlap we also generated overlaps based on the actual MR overlap data (see Section 5.1).

We repeated this procedure to generate 100 synthetic data sets, and estimated parameters for the correct model with these data sets. We checked the sensitivity to initialization method and to incorrectly specified monthly efforts. Finally, we used our algorithm to fit models on these data when we misspecify the model by omitting an important variable or by adding an unnecessary variable.

(The reader may find it strange that we simulate exponential data, which have constant coefficient of variation, and model these data as if they were Poisson, which have constant variance to mean ratio. This is intentional; we believe from other studies that the true data have constant coefficient of variation, but the larger changes are not only more variable, but also more important to model correctly. We believe coefficients from a Gamma model are too dependent on the smaller observations.)

5.1 MR overlap distribution

Overlap between MRs is an important component of the software engineering problem. One third of actual MRs have the property that they are closed after a subsequent MR is opened. To produce a realistic simulation, we had to reproduce MR overlap. Number the MRs for a single developer in order of their open times O_i so that for all i, $O_{i+1} \geq O_i$. Let C_i be the closing time of MR i. The overlap is defined as follows:

$$OVL_i \stackrel{\text{def}}{=} \begin{cases} O_{i+1} - C_i & \text{if} \quad O_{i+1} > C_i \\ \frac{C_i - O_{i+1}}{C_i - O_i} & \text{if} \quad O_{i+1} \leq C_i \end{cases} \tag{8}$$

The actual overlap when $O_{i+1} \leq C_i$ has to be defined relative to the opening time of the previous MR to preserve MR ordering. The skips ($O_{i+1} > C_i$) may be defined in absolute time. The empirical distribution of skips and overlap is in Figure 1. The probability of overlap in actual data was $P(O_{i+1} \leq C_i) = 0.3$. The overlap closely followed a Beta($\alpha = 0.85, \beta = 0.25$) distribution (with pdf $\{\Gamma(\alpha)\Gamma(\beta)\}^{-1}\Gamma(\alpha+\beta)x^{\alpha-1}(1-x)^{\beta-1}$), while negative skips ($OVL_i : O_{i+1} > C_i$) followed a Weibull($\gamma = 0.5, \beta = 2.9$) distribution (with pdf $\gamma\beta^{-1}x^{\gamma-1}\exp(-\beta^{-1}x^{\gamma})$).

The overlap distribution indicates that when the overlap occurs, MRs are likely to be started at almost the same time. The skip distribution is fairly heavy tailed which is probably caused by vacation or other developer's activities not directly related to changing the code.

5.2 Estimates with the correct model

In all of the following the hundred data sets were simulated from the model in Equation (6) with coefficients as in Table 1. Overlap of MRs was generated as described in Section 5.1. The algorithm in all cases is run for thirty iterations for each data set.

In the simulated data (more so than in practice) the monthly efforts vary across months. While in some situations the true monthly effort information is available (see, e.g., Graves and Mockus, 1998), this situation is not the norm. In the simulation we have the advantage of knowing the monthly MR efforts exactly; in the next section we test the effect of not knowing this information. Table 3 shows the results. As expected, the results are very close to the actual values. The only exceptions are coefficients for the rarely occurring cleanup and inspection MRs. Those two coefficients appear consistently below the values used in the simulation.

Typically such behavior would not cause a problem in practice (here, the cleanup and inspection coefficients were not significantly different from the new coefficient), because the uncertainty estimates will be large for coefficients estimated on the basis of a small number of such MRs. There were 15778 inspection and 30538 cleanup MRs (4.5% and 8.7% percent respectively) out of total 348204 MRs in the generated 100 samples. However, we recommend against using predictors that occur rarely in the data and which might be correlated with individual developers. Confounding helps explain the bias problem in this example, because the developer with the least productive coefficient has the smallest percentage, and the most productive developer has the highest percentage of inspection and cleanup MRs.

5.3 Conservativeness of MAIMED when $\boldsymbol{Y_{\cdot jd} \equiv 1}$

In all the remaining simulations, we set $Y_{\cdot jd}$ to one, to reflect frequent situations in practice when the actual monthly effort is not available or unreliable. Not surprisingly, we find that the resulting estimated coefficients are biased toward zero. In this case MAIMED errs on the conservative side, tending to suggest that too few factors make significant contributions to effort. In particular, the NOT coefficient is underestimated, suggesting that if the exact monthly efforts were known, the effect of the tool usage would have been even larger. (This conservativeness will generally be less pronounced in practice than in this simulation, since actual monthly efforts are close to one.) Table 4 shows the means of the estimated coefficients and their standard deviations.

5.4 Sensitivity to initialization method

Next we explore the sensitivity of MAIMED to alternative initialization methods. The first initialization method in Equation (2) divides monthly effort evenly among MRs open that month. We also used an alternative initialization method which divides monthly effort proportionally to the time each MR is open during that month:

$$Y_{ijd}(0) = \frac{\text{Interval}_{ijd}}{\sum_i \text{Interval}_{ijd}} Y_{\cdot jd}, \tag{9}$$

where Interval_{ijd} denotes the time MR i (done by developer d) was open during month j. Another initialization method is usable only with simulated data, where we know the monthly MR efforts exactly:

$$Y_{ijd}(0) = \text{Interval}_{ijd}. \tag{10}$$

The total monthly efforts are then no longer unity but rather $Y_{\cdot jd}^{\text{exact}} = \sum_i \text{Interval}_{ijd}$. In this case the initialization provides actual simulated effort values for each MR.

The results (not shown) are identical (up to two significant digits) to the results in Table 4, i.e., the initialization method does not affect the values of the estimated coefficients. In iterative proportional fitting, the initial values for the cells can be used to determine the association structure between the rows and columns. Since the choice of initial values does not have a strong effect on the results, MAIMED does not make, or need to make, strong assumptions about this association structure.

5.5 Fitting a misspecified model

To test the behavior of the estimator under model misspecification, we fitted the model omitting the size predictor and a set of models with correlated predictors.

Table 5 shows the estimates obtained by omitting the size predictor. The estimates are not significantly different from the ones obtained using the correct model in Table 4.

We also ran a sequence of tests by adding a coefficient correlated with the size covariate. Since the logarithm of the size was used in the model, the collinear covariate was correlated with the the logarithm of size. Medium correlation of 0.5 and high correlations of 0.8 and 0.95 were used. The results are in Table 6.

The results are as expected — the collinear coefficient is not significantly different from zero and the model coefficients are not significantly different from those in Table 4. The collinear predictor increases uncertainty of the size coefficient, but does not affect the other estimates.

6. CONCLUSIONS

This paper introduced a method for quantifying the extent to which properties of changes to software affect the effort necessary to implement them. The problem is difficult because measurements of the response variable, effort, are available only at an aggregated level. To solve this difficulty we propose an algorithm which imputes disaggregated values of effort, which can be improved by using a statistical model. The method may be applied to the problem of estimating a model for the row sums given only the column sums and zero pattern of an arbitrary sparse non-negative table. The method turns out to be an instance of the EM algorithm.

We provided three examples of data analyses in which we used the new algorithm to address important questions in software engineering: monitoring degradation of the code as manifested by increased effort, quantifying the benefits of a development tool, and identifying subsystems of code which are especially difficult.

We used simulations to explore the performance of the algorithm on data similar to that observed in practice. The simulations demonstrated that given enough data, the algorithm can recover the parameters of a true model, and that the algorithm still performs well when an important variable is left out of the analysis or when an extra variable correlated with one of the true predictors is included in the model. Also, the algorithm is conservative in the frequent practical situation in which aggregated effort values must be assumed to be one unit of effort per month.

We describe, in Appendix 1, how to use software which implements the algorithm (it is available for download) for the benefit of practitioners. We also briefly discuss the amount of time it takes to run the algorithm.

APPENDIX 1: SOFTWARE

Although the algorithm is fairly simple to implement, we describe a set of functions in S-PLUS that would help practitioners use the methodology more readily. The code contains five S-PLUS functions:

maimed: the main wrapper function which prepares the data objects for initialization and fitting. The main parameters are:

> *data*: the data frame containing a list of the MRs with required fields: “name”— developer name, “open” and “close” — date specified in fractional months (the first MR must have value 0 in the “open” field), as well as additional covariates as needed;
>
> *formula*: the vector of predictor names to be used when performing the fitting. The predictors are taken from the the dataframe *data*. The first predictor should always be “name” since differences among developers represent the largest variation in the effort required to complete an MR;
>
> *effmat*: optional reported effort matrix where rows correspond to distinct developer names and columns to months. If the matrix is not provided it is assumed to contain unit efforts in each cell;
>
> *calculateEffort*: boolean variable indicating whether to calculate the effort matrix based on the intervals of the MRs as in Equation (10);
>
> *weighted*: the boolean variable indicating whether to initialize the algorithm weighting MRs by their open time during a month as in Equation (9);
>
> *jk*: the boolean variable indicating whether or not to perform jackknife estimation of errors;

maimed.validate: computes standard deviations and t-statistics from the jackknife estimates obtained with *maimed*;

maimed.initial: default initialization function which distributes effort equally among MRs open in a particular month;

maimed.initial2: alternative initialization function which distributes effort proportionally to the length of time MRs are open;

maimed.initial3: initialization function which calculates monthly effort by adding lengths of time open for each MR in a month;

maimed.fit: runs the fitting algorithm.

Following is a detailed example analysis with some practical considerations. First, we create a data frame *data* using information retrieved from change management database using, for example, the *SoftChange* system (Mockus *et al*, 1999). Each MR should occupy a row in this data frame with the following fields: *name* containing the developer's name or login; *bug*, containing 1 if the change is bug fix (otherwise 0); *clean* containing 1 if the change is perfective; *new* containing 1 if the change is adaptive; *ndelta* containing size of the change in number of deltas; *open* and *close* representing timestamp of the first and the last delta (or open and close time for an MR) in unix time format (seconds since 1970); and additional factors *ToolA* and *ToolB* that indicates whether tool A or/and B were used to complete this MR.

First we select only MRs done by developers that completed at least 20 MRs with and without the tool. This might be done in following way:

```
usage.table <- table(data$name,data$usedX);
name.subset <- table(data$name)[usage.table[,1]>20 & usage.table[,2]>20];
data.new <- data[match(data$name, name.subset, nomatch=0)>0,];
```

Now the dataframe *data.new* contains only developers that worked with and without the tool and hence we will be able to make sharper comparisons. In the next step we transform the *open* and *close* fields into fractional months starting at zero:

```
data.new$open  <- data.new$open/3600/2/365.25;
data.new$close <- data.new$close/3600/2/365.25 - min(data.new$open);
data.new$open  <- data.new$open - min(data.new$open);
```

We may also transform the size of the change by a logarithm:

```
data.new$logndelta <- log(data.new$ndelta+1);
```

Finally we may run the *maimed* algorithm to estimate the coefficients for several models. One of the models might be:

$$E(Y_{i\cdot d}) = \#delta^{\alpha_1} \times \prod_{\text{types } t} (\beta_t^{I\{Type(i,d)=t\}}) \times \prod_{\text{tools } k} \gamma_k^{I\{Tool(i,d)=k\}}.$$

To fit the model we run:

```
model <- maimed (data.new, formula=c("name","logndelta","bug","clean","ToolA","ToolB"))
```

Notice that since in this example the categorization had three types of changes (bug, clean, new) we included only first two in the model. The remaining type (new) is used the basis for comparison with the first two. If all the indicators were included we would get an over determined model.

Once we find a subset of interesting models we estimate the significance of the coefficients via jackknife:

```
model <- maimed (data.new, formula=c("name","logndelta","bug","clean","ToolA","ToolB"),jk=T)
```

The result contains $n+1$ estimates of coefficients obtained by running maimed algorithm on the full dataset and by removing one of the N different developers. The significance values may be calculated as follows:

```
maimed.validate(model.jk)
```

Sample output from this command follows:

	estimate	stdev	t-statistic	p-value
lognumdelta	0.69	0.12	5.57	2.4e-05
bug	0.64	0.18	3.49	2.6e-03
clean	-0.33	0.58	-0.56	5.8e-01
ToolA	-1.40	0.19	-7.35	8.0e-07
ToolB	0.03	0.17	0.18	8.6e-01
Developer1	-2.81	0.21	-13.40	8.3e-11
Developer2	-2.08	0.18	-11.49	1.0e-09
Developer3	-2.86	0.11	-25.78	1.1e-15
Developer4	-2.86	0.17	-16.57	2.4e-12
Developer5	-2.85	0.21	-13.35	9.0e-11
Developer6	-2.58	0.19	-13.58	6.7e-11
Developer7	-2.64	0.21	-12.31	3.3e-10

We see that bug fixes are significantly harder than new code, and tool A significantly reduces the change effort. Cleanup (perfective) changes are not significantly easier than new code chnages and Tool B is not significantly increasing change effort. Hence for practical purposes the model may be rewritten using the estimates above as

$$E(\text{effort}) \quad = \quad \#delta^{0.69} \times e^{0.63I(bug)} \times e^{-1.41I(ToolA)}.$$

Here I(characteristic) is equal to 1 if the change has that characteristic, and otherwise equals 0. This implies that, for example, changes with 18 delta are 50 percent harder than changes with 10 delta ($18^{0.69}/10^{0.69} = 1.5$), bug fixes are 88 percent harder than new code changes ($\exp(0.63) = 1.88$), and tool A reduces change effort more than four times ($\exp(-1.41) = 0.24$).

The constant multiplier is included with all developer coefficients. The ratio of the highest and the lowest coefficient is around two: $e^{-2.08}/e^{-2.86} = 2.18$.

The time it takes to obtain estimates is moderate. All processing for nine developers, 4014 changes, and 86 months took 10 minutes on a Pentium II™computer using S-PLUS 4.0 for Windows. This included 10 runs of the algorithm to perform the jackknife procedure.

APPENDIX 2: FITTING POISSON MODELS TO GAMMA DATA

Here is a heuristic argument in favor of using likelihood functions which are different than those that we believe the data came from. Consider estimation using a (e.g. generalized linear) model $\theta_i = g_i(\beta)$ with log likelihood $\theta_i y_i - \psi(\theta_i)$ from the exponential family ($1 \leq i \leq N$). (For notational simplicity suppose β is not vector-valued). Denote $(\partial/\partial\beta)\theta_i$ by b_i (for example, if $\theta_i = \exp(\beta x_i)$, then $b_i = x_i \exp(\beta x_i)$). Recall that the mean $\mu_i = \psi'(\theta_i)$ and the variance $V_i = \psi_i''(\theta_i)$. Maximizing $\sum_{i=1}^N \{\theta_i y_i - \psi(\theta_i)\}$ with respect to β gives the equation $0 = \sum_{i=1}^N b_i(y_i - \mu_i)$. Now consider the problem of minimizing $\sum_{i=1}^N w_i(y_i - \mu_i)^2$ for positive constants w_i. Differentiation with respect to β gives the equation $0 = \sum_{i=1}^N w_i V_i^{-1} b_i(y_i - \mu_i)$. Since V_i in general depends on β, these two problems cannot quite be made equivalent by setting $w_i = V_i^{-1}$, unless the V_i's are available externally. However, we can conclude that gamma GLM's are closely related to the problem of minimizing $\sum_{i=1}^N \mu_i^{-2}(y_i - \mu_i)^2$, while Poisson GLM's are related to the problem of minimizing $\sum_{i=1}^N \mu_i^{-1}(y_i - \mu_i)^2$. The first minimization problem is unsatisfactory, because it penalizes 50% errors as strongly for small μ_i as for large μ_i, while in our problem, multiplicative errors are more serious for large μ_i because they correspond to larger amounts of effort. Roughly speaking, the Poisson model penalizes errors proportionally to their magnitude. This argument is meant to illustrate the strengths and weaknesses of various error distributions rather than to advocate using objective functions which are not likelihoods.

ACKNOWLEDGMENTS

The two anonymous referees for this paper were remarkably helpful, to the extent that a mention in the acknowledgments of the paper is probably inadequate to express our thanks. In particular they are responsible for the identification of MAIMED as an EM algorithm. We thank George Schmidt, Janis Sharpless, Harvey Siy, Mark Ardis, David Weiss, Alan Karr, and Iris Dowden, for their help and valuable suggestions. This research was supported in part by NSF grants SBR-9529926 and DMS-9208758 to the National Institute of Statistical Sciences. The work was performed when Dr. Graves was with the National Institute of Statistical Sciences and Bell Laboratories.

NOTES

1. By doing this, one can avoid modeling the marked point process according to which the need for work comes in, and the manner in which changes which need to be implemented are assigned to developers by their managers.

2. We strictly speaking do not assume a Poisson distribution because effort values need not be integers. The only critical part of the Poisson assumption is that the variance of a random effort is proportional to its mean.

REFERENCES

D. Atkins, T. Ball, T. Graves, and A. Mockus, "Using version control data to evaluate the effectiveness of software tools," in *1999 International Conference on Software Engineering*, (Los Angeles, CA), May 1999.

L. A. Belady and M. M. Lehman, "A model of large program development," *IBM Systems Journal*, pp. 225–252, 1976.

B. Boehm, *Software Engineering Economics.* Prentice-Hall, 1981.

B. Curtis, "Substantiating programmer variability," *Proceedings of the IEEE*, vol. 69, p. 846, July 1981.

J. N. Darroch and D. Ratcliff, "Generalized iterative scaling for log-linear models," *Annals of Mathematical Statistics*, vol. 43, pp. 1470–1480, 1972.

W. E. Deming and F. F. Stephan, "On a least-squares adjustment of a sampled frequency table when the expected marginal totals are known," *Annals of Mathematical Statistics*, vol. 11, pp. 427–444, 1940.

B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans.* Philadelphia, PA: Society for Industrial and Applied Mathematics, 1982.

B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap.* New York: Chapman and Hall, 1993.

S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? assessing the evidence from change management data," *IEEE Trans. Soft. Engrg.*, 2000. To appear.

T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Trans. Soft. Engrg.*, vol. 26, pp. 1–9, 2000.

T. L. Graves and A. Mockus, "Inferring change effort from configuration management data," in *Metrics 98: Fifth International Symposium on Software Metrics*, (Bethesda, Maryland), pp. 267–273, November 1998.

S. Karlin and H. M. Taylor, *A First Course in Stochastic Processes, 2nd Edition.* New York: Academic Press, 1975.

K. Martersteck and A. Spencer, "Introduction to the 5ESS(TM) switching system," *AT&T Technical Journal*, vol. 64, pp. 1305–1314, July-August 1985.

P. McCullagh and J. A. Nelder, *Generalized Linear Models, 2nd ed.* New York: Chapman and Hall, 1989.

G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions.* : New York, John Wiley and Sons, Inc., 1997.

A. K. Midha, "Software configuration management for the 21st century," *Bell Labs Technical Journal*, vol. 2, no. 1, Winter 1997.

A. Mockus, S. G. Eick, T. Graves, and A. F. Karr, "On measurement and analysis of software changes," tech. rep., Bell Labs, Lucent Technologies, 2000.

A. Mockus and L. G. Votta, "Identifying reasons for software changes using historic databases," International Conference on Software Maintenance, San Jose, California, October 2000, to appear.

D. L. Parnas, "Software aging," in *Proceedings 16th International Conference On Software Engineering*, (Los Alamitos, California), pp. 279–

287, IEEE Computer Society Press, 16 May 1994.

M. Rochkind, "The source code control system," *IEEE Trans. on Software Engineering*, vol. 1, no. 4, pp. 364–370, 1975.

D. B. Rubin, *Multiple Imputation for Nonresponse in Surveys,* New York: John Willey, 1987.

D. B. Rubin, "Multiple imputations after 18+ years," *JASA*, vol. 91, pp. 473–480, 1996.

G. Seber, *Linear Regression Analysis.* New York: Wiley, 1977.

E. B. Swanson, "The dimensions of maintenance," in *2nd Conf. on Software Engineering*, (San Francisco, California), pp. 492–497, 1976.

M. A. Tanner, *Tools for Statistical Inference.* New York, Berlin, Heidelberg: Springer-Verlag, 1993.

Y. Vardi, "Network tomography: Estimating source destination traffic intensities from link data," *JASA*, vol. 91, no. 433, pp. 365–377, 1996.

Table 1. Estimated coefficients and their precision. By definition, $\log \beta_{NEW} = \log \theta_{USED} = 0$.

	γ	$\log \beta_{BUG}$	$\log \beta_{CLEANUP}$	$\log \beta_{INSPECT}$	$\log \theta_{NOT}$	$\log \theta_{CONTROL}$
estimate	0.17	0.31	-0.34	-0.32	0.29	-0.11
std err	0.14	0.11	0.53	0.53	0.19	0.21

Table 2. Estimated coefficients and precision for the code base effects.

	γ	γ_1	$\log \beta_{BUG}$	$\log \beta_{CLEANUP}$	$\log \beta_{INSPECT}$	$\log \theta_{HIGH}$
estimate	0.08	0.16	-0.107	0.233	0.204	0.249
std err	0.06	0.07	0.016	0.03	0.08	0.07

NOTE: By definition, $\log \beta_{NEW} = \log \theta_{LOW} = 0$.

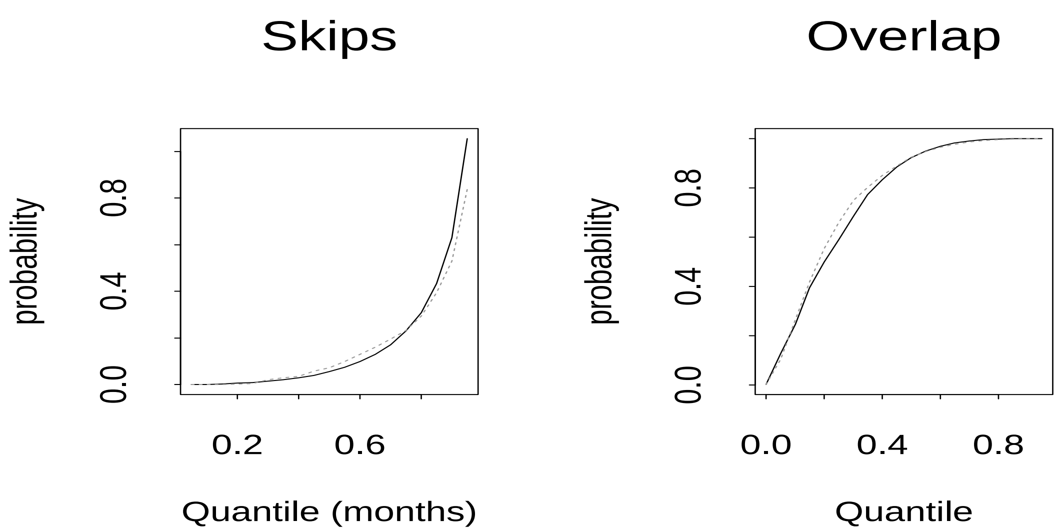

Figure 1. Empirical distribution of MR overlap. The dotted line shows actual distribution and the solid line shows parametric distribution used to simulate the data.

Table 3. Estimated coefficients for correct model with correct monthly efforts.

	γ	$\log \beta_{BUG}$	$\log \beta_{CLEANUP}$	$\log \beta_{INSPECT}$	$\log \theta_{NOT}$	$\log \theta_{CONTROL}$
true	0.17	0.31	-0.34	-0.32	0.29	-0.11
mean	0.18	0.34	-0.61	-0.59	0.31	-0.10
stdev	0.079	0.18	0.64	0.87	0.19	0.23
t-value	0.77	1.64	-4.21	-3.05	1.05	0.32
jk stderr	0.14	0.11	0.53	0.53	0.19	0.21

NOTE: t-values (for testing the hypothesis that the mean estimated coefficients from simulation are equal to their true values) are obtained by subtracting the coefficients used in simulation. The final row lists the standard errors obtained by the jackknife for comparison to these standard deviations.

Table 4. Estimated coefficients for correct model with incorrect monthly efforts. All coefficients are rounded to the two significant digits.

	γ	$\log \beta_{BUG}$	$\log \beta_{CLEANUP}$	$\log \beta_{INSPECT}$	$\log \theta_{NOT}$	$\log \theta_{CONTROL}$
true	0.17	0.31	-0.34	-0.32	0.29	-0.11
mean	0.087	0.17	-0.15	-0.22	0.16	-0.043
stdev	0.049	0.075	0.18	0.28	0.11	0.097

Table 5. Estimated coefficients for model with size coefficient omitted and with incorrect monthly efforts.

	γ	$\log\beta_{BUG}$	$\log\beta_{CLEANUP}$	$\log\beta_{INSPECT}$	$\log\theta_{NOT}$	$\log\theta_{CONTROL}$
mean	—	0.15	-0.15	-0.20	0.16	-0.067
stdev	—	0.075	0.18	0.27	0.11	0.095

Table 6. Estimated coefficients for incorrect model.

	γ	extra	$\log\beta_{BUG}$	$\log\beta_{CLEANUP}$	$\log\beta_{INSPECT}$	$\log\theta_{NOT}$	$\log\theta_{CONTROL}$
E(0.5)	0.091	-0.0055	0.167	-0.15	-0.22	0.15	-0.043
Stdev(0.5)	0.056	0.041	0.075	0.18	0.28	0.11	0.097
E(0.8)	0.085	0.0021	0.17	-0.15	-0.22	0.16	-0.043
Stdev(0.8)	0.082	0.058	0.076	0.18	0.28	0.11	0.097
E(0.95)	0.091	-0.0035	0.17	-0.15	-0.22	0.15	-0.044
Stdev(0.95)	0.15	0.11	0.075	0.18	0.28	0.11	0.097

NOTE: The predictor is collinear with $\log(size+1)$ with incorrect monthly efforts. The row labels $E(\rho)$ refer to the mean estimated coefficient when estimating a model with an extra unimportant covariate which has correlation ρ with $\log(1+\text{Size})$. Stdev(ρ) is the standard deviation of these coefficients.